06 | 分片机制：为什么说Range是更好的分片策略？

2020-08-21 王磊

分布式数据库30讲

进入课程 >

讲述：王磊

时长 18:25　大小 16.87M

你好，我是王磊，你也可以叫我 Ivan。

在这一讲的开头，我想请你思考一个问题，你觉得在大规模的业务应用下，单体数据库遇到的主要问题是什么？对，首先就是写入性能不足，这个我们在第 4 讲也说过，另外还有存储方面的限制。而分片就是解决性能和存储这两个问题的关键设计，甚至不仅是分布式数据库，在所有分布式存储系统中，分片这种设计都是广泛存在的。

所以今天，就让我们好好了解一下，分片到底是怎么回事儿。

什么是分片

分片在不同系统中有各自的别名，Spanner 和 YugabyteDB 中被称为 Tablet，在 HBase 和 TiDB 中被称为 Region，在 CockraochDB 中被称为 Range。无论叫什么，概念都是一样的，分片是一种水平切分数据表的方式，它是数据记录的集合，也是数据表的组成单位。

分布式数据库的分片与单体数据库的分区非常相似，区别在于：分区虽然可以将数据表按照策略切分成多个数据文件，但这些文件仍然存储在单节点上；而分片则可以进一步根据特定规则将切分好的文件分布到多个节点上，从而实现更强大的存储和计算能力。

分片机制通常有两点值得关注：

1. 分片策略

主要有 Hash（哈希）和 Range（范围）两种。你可能还听到过 Key 和 List，其实 Key 和 List 可以看作是 Hash 和 Range 的特殊情况，因为机制类似，我们这里就不再细分了。

2. 分片的调度机制

分为静态与动态两种。静态意味着分片在节点上的分布基本是固定的，即使移动也需要人工的介入；动态则是指通过调度管理器基于算法在各节点之间自动地移动分片。

我把分片机制的两个要点与第 5 讲提到的两种架构风格对应了一下，放到下面的表格中，希望能给你带来更直观的感受。

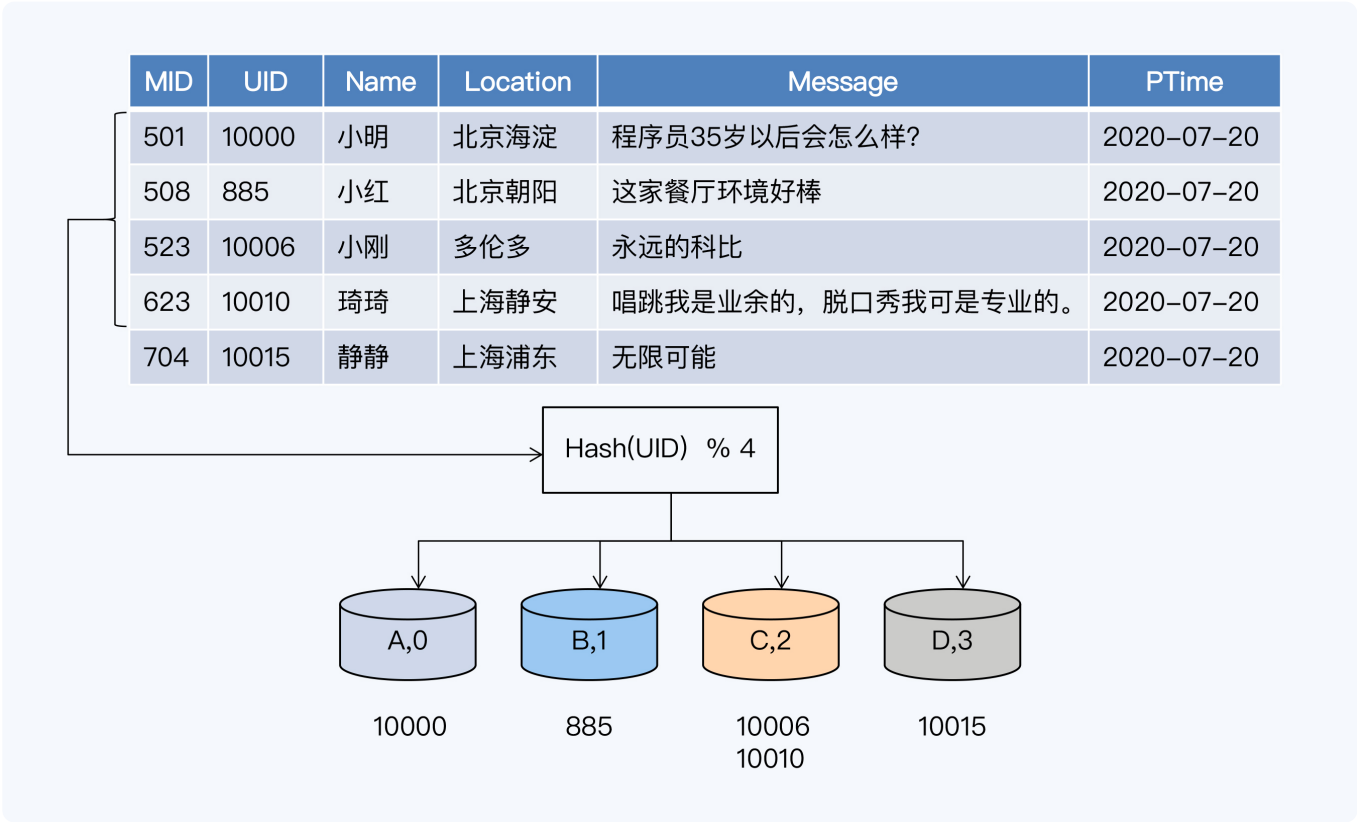
	静态	动态
Hash	PGXC / NewSQL	N/A
Range	PGXC	NewSQL

从表格中可以看出，PGXC 只支持静态的 Hash 分片和 Range 分片，实现机制较为简单，所以，我就从这里开始展开吧。

PGXC

Hash 分片

Hash 分片，就是按照数据记录中指定关键字的 Hash 值将数据记录映射到不同的分片中。我画了一张图来表示 Hash 分片的过程。

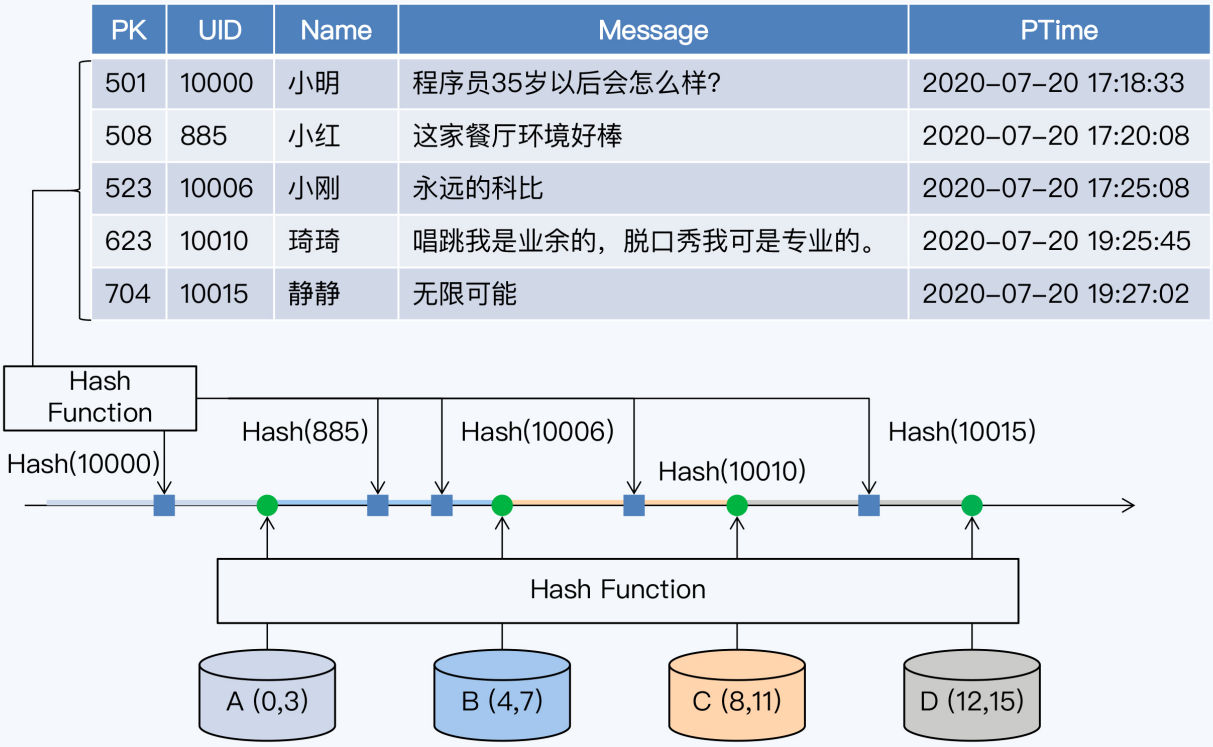


图中的表格部分显示了一个社交网站的记录表，包括主键、用户 ID、分享内容和分享时间等字段。假设以用户 ID 作为关键字进行分片，系统会通过一个 Hash 函数计算用户 ID 的 Hash 值而后取模，分配到对应的分片。模为 4 的原因是系统一共有四个节点，每个节点作为一个分片。

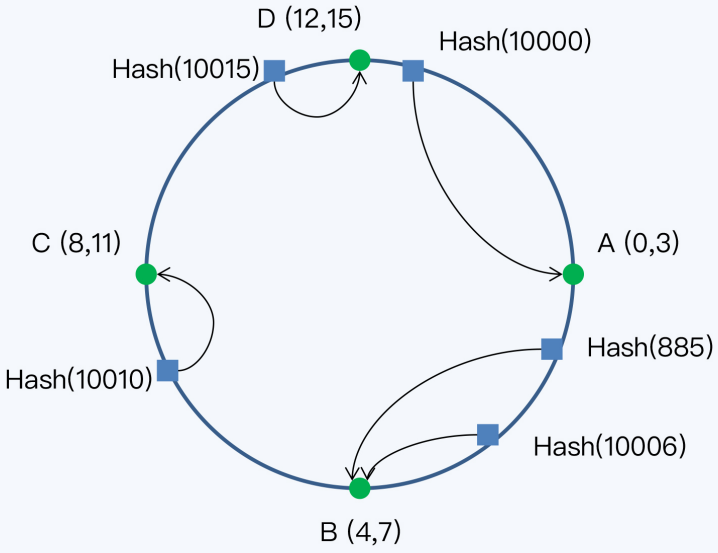
因为 Hash 计算会过滤掉数据原有的业务特性，所以可以保证数据非常均匀地分布到多个分片上，这是 Hash 分片最大的优势，而且它的实现也很简洁。但示例中采用的分片方法直接用节点数作为模，如果系统节点数量变动，模也随之改变，数据就要重新 Hash 计算，从而带来大规模的数据迁移。显然，这种方式对于扩展性是非常不友好的。

那接下来的问题就是，我们需要找一个方法提升系统的扩展性。你可能猜到了，这就是一致性 Hash，该算法首次提出是在论文“Consistent Hashing and Random Trees : Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web”当中。

要在工业实践中应用一致性 Hash 算法，首先会引入虚拟节点，每个虚拟节点就是一个分片。为了便于说明，我们在这个案例中将分片数量设定为 16。但实际上，因为分片数量决定了集群的最大规模，所以它通常会远大于初始集群节点数。

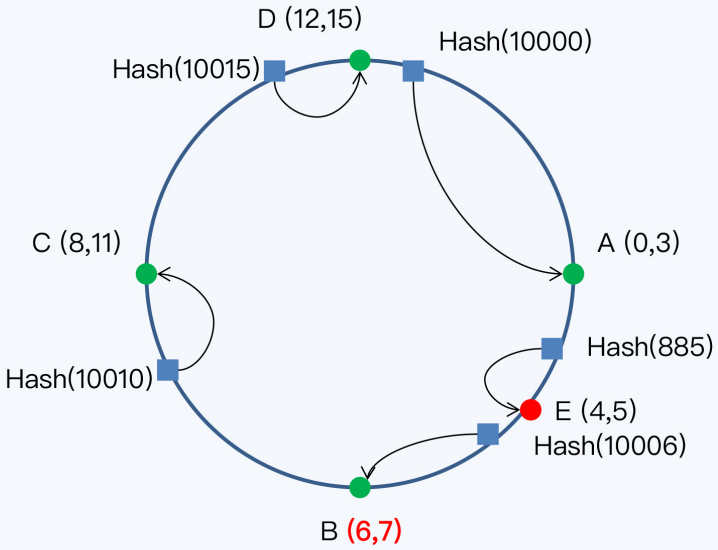


16 个分片构成了整个 Hash 空间，数据记录的主键和节点都要通过 Hash 函数映射到这个空间。这个 Hash 空间是一个 Hash 环。我们换一种方式画图，可以看得更清楚些。



节点和数据都通过 Hash 函数映射到 Hash 环上，数据按照顺时针找到最近的节点。

当我们新增一台服务器，即节点 E 时，受影响的数据仅仅是新服务器到其环空间中前一台服务器（即沿着逆时针方向的第一台服务器）之间数据。结合我们的示例，只有小红分享的消息从节点 B 被移动到节点 E，其他节点的数据保持不变。此后，节点 B 只存储 Hash 值 6 和 7 的消息，节点 E 存储 Hash 值 4 和 5 的消息。



Hash 函数的优点是数据可以较为均匀地分配到各节点，并发写入性能更好。

本质上，Hash 分片是一种静态分片方式，必须在设计之初约定分片的最大规模。同时，因为 Hash 函数已经过滤掉了业务属性，也很难解决访问业务热点问题。所谓业务热点，就是由于局部的业务活跃度较高，形成系统访问上的热点。这种情况普遍存在于各类应用中，比如电商网站的某个商品卖得比较好，或者外卖网站的某个饭店接单比较多，或者某个银行网点的客户业务量比较大等等。

Range 静态分片

与 Hash 分片不同，Range 分片的特点恰恰是能够加入对于业务的预估。例如，我们用“Location”作为关键字进行分片时，不是以统一的行政级别为标准。因为注册地在北京、上海的用户更多，所以这两个区域可以按照区县设置分片，而海外用户较少，可以按国家设置为分片。这样，分片间的数据更加平衡。

MID	UID	Name	Location	Message	PTime
501	10000	小明	北京海淀	程序员35岁以后会怎么样?	2020-07-20
508	885	小红	北京朝阳	这家餐厅环境好棒	2020-07-20
523	10006	小刚	加拿大	永远的科比	2020-07-20
623	10010	琦琦	上海静安	唱跳我是业余的，脱口秀我可是专业的。	2020-07-20
704	10015	静静	上海浦东	无限可能	2020-07-20

但是，这种方式依然是静态的，如果海外业务迅速增长，服务海外用户的分片将承担更大的压力，可能导致性能下降，用户体验不佳。

相对 Hash 分片，Range 分片的适用范围更加广泛。其中一个非常重要的原因是，Range 分片可以更高效地扫描数据记录，而 Hash 分片由于数据被打散，扫描操作的 I/O 开销更大。但是，PGXC 的 Range 分片受限于单体数据库的实现机制，很难随数据变动和负载变化而调整。

虽然有些 PGXC 同时支持两种分片方式，但 Hash 分片仍是主流，比如 GoldenDB 默认使用 Hash 分片，而 TBase 仅支持 Hash 分片。

NewSQL

总体上，NewSQL 也是支持 Hash 和 Range 两种分片方式的。具体就产品来说，CockroachDB 和 YugabyteDB 同时支持两种方式，TiDB 仅支持 Range 分片。

NewSQL 数据库的 Hash 分片也是静态的，所以与 PGXC 差别不大，这里就不再赘述了。接下来，我们重点学习下 Range 动态分片。

Range 动态分片

NewSQL 的 Range 分片，多数是用主键作为关键字来分片的，当然主键可以是系统自动生成的，也可以是用户指定的。既然提供了用户指定主键的方式，那么理论上可以通过设定主键的产生规则，控制数据流向哪个分片。但是，主键必须保证唯一性，甚至是单调递增的，导致这种控制就会比较复杂，使用成本较高。所以，我们基本可以认为，分片是一个系统自动处理的过程，用户是感知不到的。这样做的好处显然是提升了系统的易用性。

我们将 NewSQL 的 Range 分片称为动态分片，主要有两个原因：

1. **分片可以自动完成分裂与合并**

当单个分片的数据量超过设定值时，分片可以一分为二，这样就可以保证每个分片的数据量较为均衡。多个数据量较少的分片，会在一定的周期内被合并为一个分片。

还是回到我们社交网站这个例子，根据消息的数量来自动分片，我们可以得到 R1、R2、R3 三个分片。

	MID	UID	Name	Location	Message	PTime
R1	501	10000	小明	北京海淀	程序员35岁以后会怎么样?	2020-07-20
	508	885	小红	北京朝阳	这家餐厅环境好棒	2020-07-20
	523	10006	小刚	加拿大	永远的科比	2020-07-20
R2	623	10010	琦琦	上海静安	唱跳我是业余的，脱口秀我可是专业的。	2020-07-20
R3	704	10015	静静	上海浦东	无限可能	2020-07-20

分片也会被均衡地调度到各个节点上，节点间的数据量也保持总体平衡。

2. **可以根据访问压力调度分片**

我们看到系统之所以尽量维持分片之间，以及节点间的数据量均衡，存储的原因外，还可以更大概率地将访问压力分散到各个节点上。但是，有少量的数据可能会成为访问热点，就是上面提到的业务热点，从而打破这种均衡。比如，琦琦和静静都是娱乐明星，有很多粉丝关注她们分享的内容，其访问量远超过普通人。这时候，系统会根据负载情况，将 R2 和 R3 分别调度到不同的节点，来均衡访问压力。

存储均衡和**访问压力均衡**，是 NewSQL 分片调度机制普遍具备的两项能力。此外，还有两项能力在Spanner 论文中被提及，但在其他产品中没有看到工程化实现。

第一是**减少分布式事务**。

对分布式数据库来说，有一个不争的事实，那就是分布式事务的开销永远不会小于单节点本地事务的开销。因此，所有分布式数据库都试图通过减少分布式事务来提升性能。

Spanner 在 Tablet，也就是 Range 分片，之下增加了目录（Directory），作为数据调度的最小单位，它的调度范围是可以跨 Tablet 的。通过调度 Directory 可以将频繁参与同样事务的数据，转移到同一个 Tablet 下，从而将分布式事务转换为本地事务。

第二是**缩短服务延时**。

对于全球化部署的分布式数据库，数据可能存储在相距很远的多个数据中心，如果用户需要访问远端机房的数据，操作延时就比较长，这受制于数据传输速度。而 Spanner 可以将 Directory 调度到靠近用户的数据中心，缩短数据传输时间。当然，这里的调度对象都是数据的主副本，跨中心的数据副本仍然存在，负责保证系统整体的高可靠性。

Directory 虽然带来新的特性，但显然也削弱了分片的原有功能，分片内的记录不再连续，扫描要付出更大成本。而减少分布式事务和靠近客户端位置这本身就是不能兼顾的，再加上存储和访问压力，分片调度机制要在四个目标间进行更复杂的权衡。

Spanner 的这种设计能达到什么样的实际效果呢？我们现在还需要继续等待和观察。

分片与高可靠的关系

高可靠是分布式数据库的重要特性，分片是数据记录的最小组织单位，也必须是高可靠的。

NewSQL 与 PGXC 的区别在于，对于 NewSQL 来说，分片是高可靠的最小单元；而对于 PGXC，分片的高可靠要依附于节点的高可靠。

NewSQL 的实现方式是复制组（Group）。在产品层面，通常由一个主副本和若干个副本组成，通过 Raft 或 Paxos 等共识算法完成数据同步，称为 Raft Group 或 Paxos Group，所以我们简称这种方式为 Group。因为不相关的数据记录会被并发操作，所以同一时刻有多个 Group 在工作。因此，NewSQL 通常支持 Multi Raft Group 或者 Multi Paxos Group。这里，我们先忽略 Multi Paxos 的另一个意思。

每个 Group 是独立运行的，只是共享相同的网络和节点资源，所以不同复制组的主副本是可以分布在不同节点的。

PGXC 的最小高可靠单元由一个主节点和多个备节点组成，我们借用 TDSQL 中的术语，将其称为 Set。一个 PGXC 是由多个 Set 组成。Set 的主备节点间复制，多数采用半同步复制，平衡可靠性和性能。这意味着，所有分片的主副本必须运行在 Set 的主节点上。

从架构设计角度看，Group 比 Set 更具优势，原因主要有两个方面。首先，Group 的高可靠单元更小，出现故障时影响的范围就更小，系统整体的可靠性就更高。其次，在主机房范围内，Group 的主副本可以在所有节点上运行，资源可以得到最大化使用，而 Set 模式下，占大多数的备节点是不提供有效服务的，资源白白浪费掉。

小结

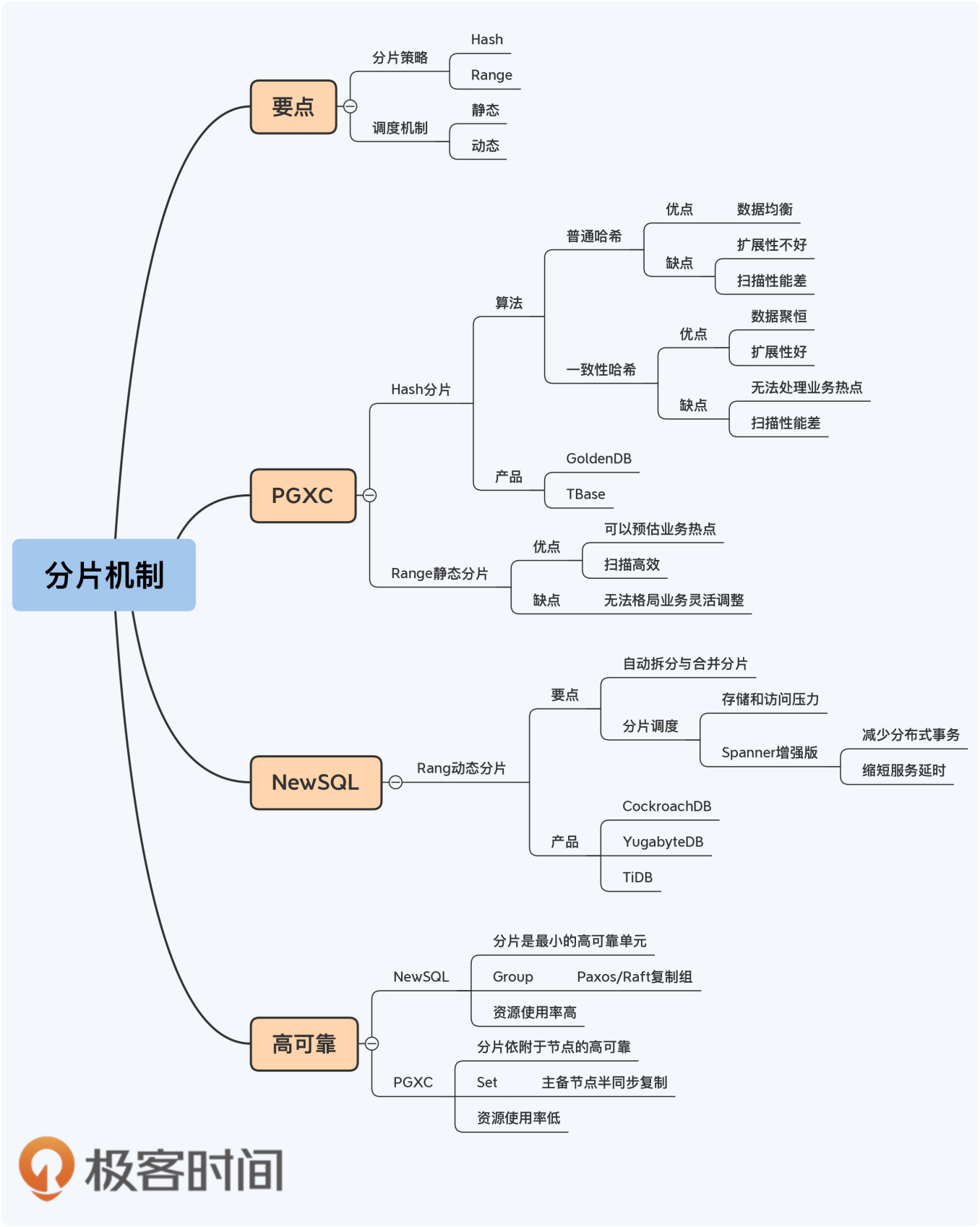
好吧，今天的内容就到这里了，我们一起回顾下这节课的重点。

1. 分片是分布式数据库的关键设计，以此实现多节点的存储和访问能力。
2. 分片机制的两个要点是分片策略和调度机制，分片策略包括 Hash 和 Range 两种，调度机制则分为静态和动态。
3. PGXC 使用单体数据库作为数据节点，往往只实现了静态分片。它的分片策略支持 Hash 和 Range 两种，其中 Hash 一般是指一致性 Hash，可以最大程度规避节点扩缩带来的影响。Hash 分片写性能出众，但查询性能差，Range 则相反。
4. NewSQL 的默认分片策略通常是 Range 分片。分片调度机制为了实现存储平衡和访问压力平衡的目标，会将分片动态调度到各个节点。Spanner 的设计又将在分片下拓展了 Directory，通过对 Directory 的调度实现减少分布式事务和缩短延时的目标，但在其他分布式数据库中尚未看到对应的实现。
5. NewSQL 架构下，分片采用 Paxos 或 Raft 算法可以构成复制组，这种复制机制相比 PGXC 的主备节点复制，提供了更高的可靠性，资源使用也更加高效。

到这里你应该已经大体了解了分布式数据库分片机制。我们说 Range 是更好的分片策略，就是因为 Range 分片有条件做到更好的动态调度，只有动态了，才能自适应各种业务场景下的数据变化，平衡存储、访问压力、分布式事务和访问链路延时等多方面的诉求。从我

个人的观点来说，NewSQL 的 Range 分片方式更加优雅，随着单体数据库底层数据同步机制的改进，未来 PGXC 可能也会向这种方式靠拢。

如果你想更深入地了解 Range 分片机制，可以研究下 BigTable 的论文。同时，因为 HBase 是业界公认的 BigTable 开源实现，所以你在它的官方文档也能找到很多有用的内容。



思考题

Range 分片的优势是动态调度，这就是说分片存储在哪个节点上是不断变化的。这时，客户端首先要知道分片的位置，就要先访问分片的元数据。你觉得这些元数据应该如何存储呢？是存储在某个中心点，还是分散在所有节点上呢？如果有多个副本，又该如何同步呢？

如果你想到了答案，又或者是触发了你对相关问题的思考，都可以在评论区和我聊聊，我会在答疑篇更系统地回复这个问题。如果你身边的朋友也对数据的分片机制，这个话题感兴趣，你也可以把今天这一讲分享给他，我们一起讨论。

学习资料

David Karge et al.: *Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web*

Fay Chang et al.: *Bigtable: A Distributed Storage System for Structured Data*

HBase: *Apache HBase ™ Reference Guide*

James C. Corbett et al.: *Spanner: Google’s Globally-Distributed Database*

提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 05 | 全局时钟：物理时钟和逻辑时钟你Pick谁?

下一篇 07 | 数据复制：为什么有时候Paxos不是最佳选择?

精选留言 (7)

写留言

真名不叫黄金

2020-08-22

猜测一下，如果是TiDB的话，将元数据存在PD，而PD本身又可部署为多节点高可用的，不过数据最终是落在etcd的，PD只是交互节点。
Spanner如何做的就不太好猜测，但是Spanner也有PD这个角色，也许是差不多的。

展开

作者回复: 是的，基本正确，点赞。另外，TiDB还做了一些优化，详细的内容在第7讲有说明

1 2

开心哥
2020-08-21

元数据搞个etcd存起来如何?

展开

作者回复: 嗯，说得对，这是方案之一，点赞

1

Geek_4e4b8b
2020-08-22

作者可以研究下mongodb的hash分片，还是挺巧妙的，他的算法依赖的应该不是节点数，而是跟数据的chunk数相关，这样增减节点，只是涉及chunk的移动，不会大面积做全部数据的重平衡。当然算法如果仅仅只是chunk数相关，那chunk数变化就会触发算法变化，所以应该是做了优化的，保证chunk的分裂只会影响分裂chunk的数据移动。

展开

南国
2020-08-22

分片数据存在分布式文件系统里，元数据像bigtable一样用一个高可靠的协调中心存，比如Zookeeper，在合并和分裂的时候修改元数据，客户端缓存需要的元数据，修改的时候通知即可

扩散性百万咸面包
2020-08-22

Hash 分片写性能出众，但查询性能差，Range 则相反。

没懂这一句话，文章中哪里有详细阐释为什么Hash分片的写性能更好呢？为什么Range的写性能就不行呢?

展开

作者回复: Hash分片会将数据比较均匀的分散在集群的各个节点上，所以性能更好，而Range的数据分布是根据编码规则（静态）或者主键（动态）的，不以追求平均分布为目标，所以性能会差些。这个性能差异问题，我在第17讲还会再说明。

游弋云端
2020-08-21

元数据集中存储，特别是能用全内存性能最好，但可靠性不足，一般做HA;或者元数据可以做一致性Hash来分片打散，个人认为Range不适合元数据，变化了数据位置不好计算。

权
2020-08-21

es的，ceph的分片机制也是类似，ceph有自动rebalance,es貌似要手动